CHAPTER 13 TREE-STRUCTURED INFORMATION IN TABLES

CONTENTS

Part Number 3201-V2.0

Revised July 1987

Contributing Author: Jonathan Sachs

Major Contributors: Larry Baer, Derry Kabcenell, Larry Stevens

RACI

CHAPTER 13 TREE-STRUCTURED INFORMATION IN TABLES

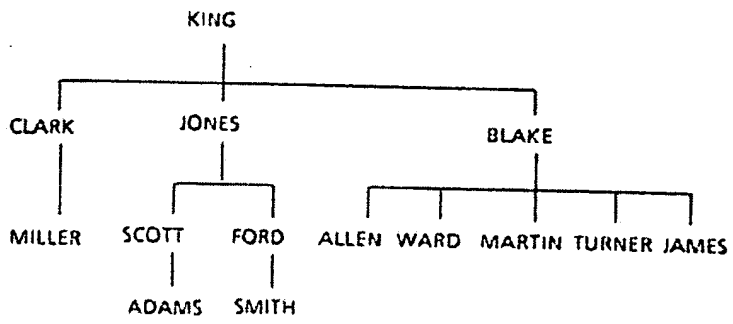
CONTENTS

ABOUT THIS CHAPTER

This chapter describes how to work with *tree structured* information in tables.

Information is "tree structured" if, in some sense, every unit of information either "belongs to" another unit, or "owns" some number of other units, or both. An organization chart like the one shown in Figure 13-1 is a common example of tree structured information. Every position on the chart either reports to (belongs to) a superior position, or is responsible for (owns) some number of subordinate positions, or both. When the organization chart is done as in Figure 13-1, it resembles an upside-down tree.

Figure 13-1
An Organization Chart



This chapter explains how to:

- define a tree structure in terms of an ORACLE table
- identify the "root" of a tree
- select a direction for a tree-structured query
- display tree-structured information from a table.

CONCEPTS OF TREE-STRUCTURED INFORMATION

For convenience, we are going to use the term "tree" to refer to a tree-structured collection of information. We will use the term "node" to identify a unit of information in a tree-structured collection. When tree-structured information is stored in an ORACLE table, each node is represented by one row in the table.

We are going to start by defining a tree as computer programmers usually define it. The definition of a tree used by SQL*Plus is actually somewhat more complex. We'll return to this in a later topic.

A tree has one node which has no owner, and which —directly or indirectly —owns all the other nodes in the tree. The ownerless node is called the *root* of the tree. In the organization chart shown in Figure 13-1, King's row is the tree's root.

If one node owns another, the owning node is said to be the *parent* of the owned node. The owned node is said to be the *child* of the owning node. Each node in a tree has exactly one parent (except for the root node, which has none). A node may have any number of children. A node with no children is called a *terminal node*, or *leaf*.

Nodes which have the same parent are called *siblings*.

A node of a tree, together with all of its children, grandchildren, and so forth, constitutes a *branch* of the tree.

Inspecting each node of a tree in turn is called *walking the tree*. In order to walk a tree, we need some systematic way of ensuring that we will inspect each node one, and only one, time. It is customary to walk a tree according to the following rules:

1. Start at the root node.
2. Inspect this node.
3. If this node has any children that have not yet been visited, move to the leftmost unvisited child and go to step 2. Otherwise:
4. If this is the root node, we're done. Otherwise:
5. Return to this node's parent and go to step 3.

If we walk the tree shown in Figure 13-1 according to these rules, we will visit the nodes in the following order: KING, CLARK, MILLER, JONES, SCOTT, ADAMS, FORD, SMITH, BLAKE, ALLEN, WARD, MARTIN, TURNER, and JAMES.

Exercise 13-1

Walk the tree shown in Figure 13-1 by applying the rules described above.

Exe

How a Tree Is Represented in a Table

The relationships that connect the nodes in a tree may be represented by relationships among the rows in a table.

For example, if the nodes shown in Figure 13-1 are represented by the rows in the table EMP, the parent-child relationships that define the tree are represented by relationships between values in the columns MGR and EMPNO. The fact that SCOTT reports to JONES (SCOTT's node is a child of JONES's node) is reflected by the fact that JONES's employee number, found in JONES's EMPNO field, is also found in SCOTT's MGR field.

Suppose we want to write a SELECT command that lists the contents of the table EMP in a way that reflects the tree structure represented in Figure 13-1. We must include a clause that defines the relationship between a parent node and a child node, and a clause that identifies the root node.

We could express the relationship that connects nodes in this tree by the following "logical expression":

parent node's EMPNO = child node's MGR

We can put this relationship into a query with the CONNECT BY clause, which looks like this:

```
CONNECT BY   PRIOR EMPNO = MGR
```

This clause says, "if one node's EMPNO equals a second node's MGR, the first node is the parent of the second." You can remember the meaning of the PRIOR operator by imagining that you are walking the tree from the root downward: you will encounter each parent node *prior* to its child.

We identify the root node with the START WITH clause, which looks like this:

```
START WITH   ENAME = 'KING'
```

This clause says, "the root node (the one to *start with*) is the node in which the logical expression ENAME = 'KING' is true."

Both the CONNECT BY and START WITH clauses must follow the FROM clause (and the WHERE clause, if present).

Exercise 13-2

To list information about employees and their managers in tree form, enter:

```
SQL> SELECT     ENAME, EMPNO, JOB, DEPTNO, MGR
  2  FROM       EMP
  3  CONNECT BY PRIOR EMPNO = MGR
  4  START WITH ENAME = 'KING';

ENAME      EMPNO JOB       DEPTNO    MGR
---------- ----- --------- ------ ------
KING        7839 PRESIDENT      10
JONES       7566 MANAGER        20   7839
FORD        7902 ANALYST        20   7566
SMITH       7369 CLERK          20   7902
SCOTT       7788 ANALYST        20   7566
ADAMS       7876 CLERK          20   7788
CLARK       7782 MANAGER        10   7839
MILLER      7934 CLERK          10   7782
BLAKE       7698 MANAGER        30   7839
WARD        7521 SALESMAN       30   7698
MARTIN      7654 SALESMAN       30   7698
TURNER      7844 SALESMAN       30   7698
JAMES       7900 CLERK          30   7698
ALLEN       7499 SALESMAN       30   7698

14 records selected.
```

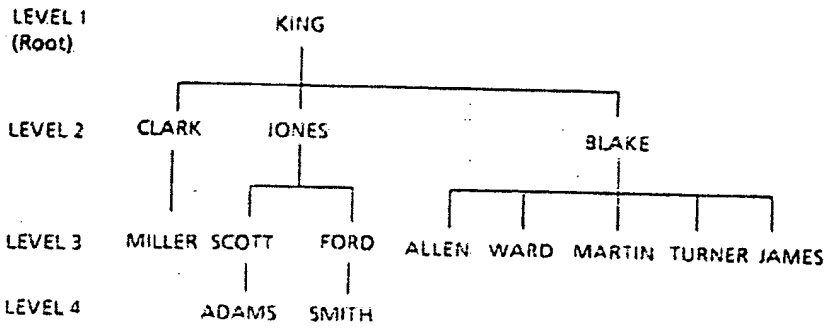
This query returns the rows of EMP by walking the tree from the specified root. King's row, the root, is returned first; then the row for Jones, who works for King; then the row for Ford, who works for Jones; and so on.

Level Numbers

The nodes in a tree may be assigned *level numbers* depending on how far removed they are from the root. The root is level 1, children of the root are level 2, grandchildren are level 3, and so forth. Figure 13-2 shows the levels of the nodes in our sample company's organization chart.

Exerci

Figure 13-2
Levels of the Organization Chart



LEVEL is a pseudo-column, like SYSDATE (introduced in Chapter 10). You may use LEVEL in an ORDER BY or GROUP BY clause, or anywhere else a column name may appear.

Exercise 13-3

To retrieve level numbers of rows in EMP, enter:

```
SQL> SELECT      LEVEL, ENAME, EMPNO, JOB, DEPTNO, MGR
  2  FROM        EMP
  3  CONNECT BY  PRIOR EMPNO = MGR
  4  START WITH  ENAME = 'KING';

LEVEL ENAME      EMPNO JOB       DEPTNO      MGR
----- ---------- ----- --------- ------- -------
    1 KING        7839 PRESIDENT      10
    2 JONES       7566 MANAGER        20    7839
    3 SCOTT       7788 ANALYST        20    7566
    4 ADAMS       7876 CLERK          20    7788
    3 FORD        7902 ANALYST        20    7566
    4 SMITH       7369 CLERK          20    7902
    2 CLARK       7782 MANAGER        10    7839
    3 MILLER      7934 CLERK          10    7782
    2 BLAKE       7698 MANAGER        30    7839
    3 ALLEN       7499 SALESMAN       30    7698
    3 WARD        7521 SALESMAN       30    7698
    3 MARTIN      7654 SALESMAN       30    7698
    3 TURNER      7844 SALESMAN       30    7698
    3 JAMES       7900 CLERK          30    7698

14 records selected.
```

Exerci

You can use the function LPAD to display the organization chart as an indented list.

Exercise 13-4

To create an indented list of employees in manager-employee order, enter:

```
SQL> COLUMN ORG_CHART FORMAT A21;
SQL> SELECT      LPAD(' ',2*LEVEL) || ENAME ORG_CHART, LEVEL, EMPNO,
  2              MGR, DEPTNO
  3  FROM        EMP
  4  CONNECT BY  PRIOR EMPNO = MGR
  5  START WITH  ENAME = 'KING';

ORG_CHART             LEVEL  EMPNO  MGR  DEPTNO
--------------------  -----  -----  ----  ------
KING                      1   7839              10
JONES                     2   7566  7839        20
SCOTT                     3   7788  7566        20
ADAMS                     4   7876  7788        20
FORD                      3   7902  7566        20
SMITH                     4   7369  7902        20
CLARK                     2   7782  7839        10
MILLER                    3   7934  7782        10
BLAKE                     2   7698  7839        30
ALLEN                     3   7499  7698        30
WARD                      3   7521  7698        30
MARTIN                    3   7654  7698        30
TURNER                    3   7844  7698        30
JAMES                     3   7900  7698        30

14 records selected.
```

In this exercise, the alias ORG_CHART gives the first column its heading. The COLUMN command makes the column 21 characters wide.

In line 1 of the SELECT command, the value of LEVEL is multiplied by 2 (2*LEVEL) to compute the number of blank spaces to precede ENAME.

You can find the average salary at each level of the employee tree by using the pseudo-column LEVEL in the GROUP BY and ORDER BY clauses:

Exercise 13-5

To find the average salary at each level of the employee tree, enter:

```
SQL> SELECT      LEVEL, AVG(SAL)
  2  FROM        EMP
  3  CONNECT BY  PRIOR EMPNO = MGR
  4  START WITH  ENAME = 'KING'
  5  GROUP BY    LEVEL
  6  ORDER BY    LEVEL;

     LEVEL  AVG(SAL)
---------- ---------
         1      5000
         2   2758.33
         3   1731.25
         4       950
```

SPECIFYING THE ROOT: THE START WITH CLAUSE

The START WITH clause specifies a condition that identifies the root of the tree: ENAME = 'KING' in the preceding exercises. Since everyone in the EMP table works either directly or indirectly for KING (the president), these queries incorporate all of the rows of the table into the organization tree.

But you can identify any other row of the table as the root, and retrieve only the branch of the tree that begins at that row. Suppose you want to know which employees work (directly or indirectly) for JONES:

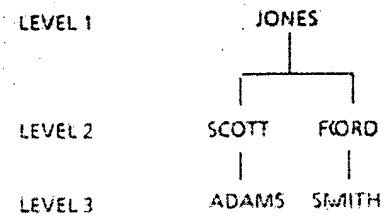
Exercise 13-6

To list information about employees who work for JONES, enter:

```
SQL> SELECT      LPAD(' ',2*LEVEL) || ENAME "ORG_CHART", LEVEL, EMPNO,
  2              MGR, DEPTNO
  3  FROM        EMP
  4  CONNECT BY  PRIOR EMPNO = MGR
  5  START WITH  ENAME = 'JONES';

ORG_CHART            LEVEL  EMPNO  MGR  DEPTNO
-------------------- -----  -----  ----  ------
JONES                    1   7566  7839      20
  SCOTT                  2   7788  7566      20
    ADAMS                3   7876  7788      20
  FORD                   2   7902  7566      20
    SMITH                3   7369  7902      20
```

Notice that the LEVEL numbers always begin with 1 at the root specified by START WITH. If you start with JONES, JONES is at level 1, as shown in Figure 13-3.

Figure 13-3
A Partial Organization Chart Starting With Jones



The START WITH clause can contain any kind of logical expression. For example, it can construct several trees at once by specifying an expression that is satisfied by more than one row, and so identifies more than one root.

Exercise 13-7

To list information about the people who work for CLARK or BLAKE, enter:

```
SQL> SELECT      LPAD(' ',2*LEVEL) || ENAME ORG_CHART, LEVEL, EMPNO,
  2              MGR, JOB, DEPTNO
  3  FROM        EMP
  4  CONNECT BY PRIOR EMPNO = MGR
  5  START WITH ENAME = 'CLARK'
  6   OR        ENAME = 'BLAKE'
  7  ORDER BY   DEPTNO;

ORG_CHART             LEVEL  EMPNO  MGR JOB       DEPTNO
-------------------- ------ ------ ---- --------- -------
CLARK                     1   7782 7839 MANAGER       10
  MILLER                  2   7934 7782 CLERK         10
BLAKE                     1   7698 7839 MANAGER       30
  MARTIN                  2   7654 7698 SALESMAN      30
  JAMES                   2   7900 7698 CLERK         30
  ALLEN                   2   7499 7698 SALESMAN      30
  TURNER                  2   7844 7698 SALESMAN      30
  WARD                    2   7521 7698 SALESMAN      30

8 records selected.
```

SELECTING A DIRECTION: THE PRIOR OPERATOR

A query may walk the tree either down from root to leaves, or up from leaves to root. You specify the direction of the walk with the PRIOR operator in the CONNECT BY clause.

In the preceding exercises, PRIOR was placed before the parent-node expression in the CONNECT BY clause, making the query walk the tree from the root down to the leaves. If you place PRIOR before the child-node expression, the query walks the tree from one or more leaves up to the root. (Note that this kind of walk will *not* visit all the nodes in the tree unless it identifies every leaf as a starting point.)

Exercise 13-8

To list information about all the people in the organization above SMITH, enter:

```
SQL> SELECT      LPAD(' ',2*LEVEL) || ENAME ORG_CHART, EMPNO, MGR,
  2              JOB, DEPTNO
  3  FROM        EMP
  4  CONNECT BY  EMPNO = PRIOR MGR
  5  START WITH  ENAME = 'SMITH';

ORG_CHART             EMPNO  MGR JOB       DEPTNO
-------------------- ------ ---- --------- -------
SMITH                  7369 7902 CLERK          20
FORD                   7902 7566 ANALYST        20
JONES                  7566 7839 MANAGER        20
KING                   7839      PRESIDENT      10
```

SELECTING ROWS

Exerci

As usual, you can use logical expressions to control which rows of a tree are displayed. When a query is tree-structured, though, you can put a logical expression in either the WHERE clause or the CONNECT BY clause.

- The WHERE clause "prunes" individual leaves of the tree.
- The CONNECT BY clause "prunes" entire branches of the tree.

Suppose you want a list of all the people who work for JONES, *except for* SCOTT.

Exercise 13-9

To list information about all the people who work for JONES except SCOTT, enter:

```
SQL> SELECT      ENAME, EMPNO, MGR, JOB, DEPTNO
  2  FROM        EMP
  3  WHERE       ENAME != 'SCOTT'
  4  CONNECT BY  PRIOR EMPNO = MGR
  5  START WITH  ENAME = 'JONES';

ENAME          EMPNO      MGR JOB         DEPTNO
---------- ---------- ------- --------- -------
JONES           7566     7839 MANAGER        20
ADAMS           7876     7788 CLERK          20
FORD            7902     7566 ANALYST        20
SMITH           7369     7902 CLERK          20
```

Figure
Effects
Expres
and CC
Clause

Since the WHERE clause contains the expression ENAME != 'SCOTT', the query eliminates SCOTT from the result. ADAMS, who works for SCOTT, is not eliminated. In other words, the WHERE clause excludes SCOTT's row, but not the branch that "hangs" from it.

To eliminate that branch entirely, use the expression in the CONNECT BY clause:

Exercise 13-10

To list information about all the employees who work for JONES, except SCOTT and those who work for SCOTT, enter:

```
SQL> SELECT      ENAME, EMPNO, MGR, JOB, DEPTNO
  2  FROM        EMP
  3  CONNECT BY  PRIOR EMPNO = MGR
  4    AND       ENAME != 'SCOTT'
  5  START WITH  ENAME = 'JONES';

ENAME      EMPNO      MGR JOB       DEPTNO
---------- ------- ------- --------- -------
JONES         7566    7839 MANAGER        20
FORD          7902    7566 ANALYST        20
SMITH         7369    7902 CLERK          20
```

You may use logical expressions in both clauses to exclude both branches and individual nodes.

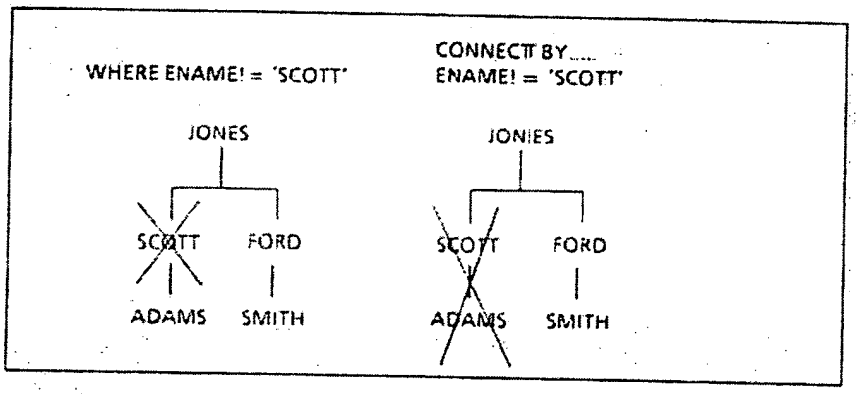


Figure 13-4
Effects of Logical Expressions in WHERE and CONNECT BY Clauses

Exercise 13-11

To list all the employees who work for JONES except FORD, SCOTT, and the people who work for SCOTT, enter:

```
SQL> SELECT      ENAME, EMPNO, MGR, JOB, DEPTNO
  2  FROM        EMP
  3  WHERE       ENAME != 'FORD'
  4  CONNECT BY  PRIOR EMPNO = MGR
  5    AND       ENAME != 'SCOTT'
  6  START WITH  ENAME = 'JONES';

ENAME      EMPNO      MGR JOB       DEPTNO
---------- ------- ------- --------- -------
JONES         7566    7839 MANAGER        20
SMITH         7369    7902 CLERK          20
```

If you specify conditions with both WHERE and CONNECT BY, the query:

1. Locates the root with the START WITH clause
2. Constructs the tree according to the relationship defined in the CONNECT BY clause
3. Walks the tree in the direction specified by PRIOR
4. Excludes branches of the tree by applying the conditions in the CONNECT BY clause
5. Excludes individual rows by applying the conditions in the WHERE clause to each row
6. Sorts the rows according to the ORDER BY clause

THE S

CHAPTER 13 TREE-STRUCTURED INFORMATION IN TABLES

CONTENTS

Part Number 3201-V2.0

Revised July 1987

Contributing Author: Jonathan Sachs

Major Contributors: Larry Baer, Derry Kabcenell, Larry Stevens

RACI

CHAPTER 13 TREE-STRUCTURED INFORMATION IN TABLES

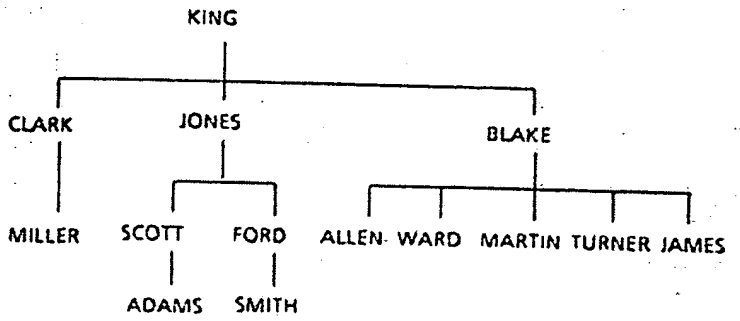
CONTENTS

ABOUT THIS CHAPTER

This chapter describes how to work with *tree structured* information in tables.

Information is "tree structured" if, in some sense, every unit of information either "belongs to" another unit, or "owns" some number of other units, or both. An organization chart like the one shown in Figure 13-1 is a common example of tree structured information. Every position on the chart either reports to (belongs to) a superior position, or is responsible for (owns) some number of subordinate positions, or both. When the organization chart is done as in Figure 13-1, it resembles an upside-down tree.

Figure 13-1
An Organization Chart



This chapter explains how to:

- define a tree structure in terms of an ORACLE table
- identify the "root" of a tree
- select a direction for a tree-structured query
- display tree-structured information from a table.

CONCEPTS OF TREE-STRUCTURED INFORMATION

For convenience, we are going to use the term "tree" to refer to a tree-structured collection of information. We will use the term "node" to identify a unit of information in a tree-structured collection. When tree-structured information is stored in an ORACLE table, each node is represented by one row in the table.

We are going to start by defining a tree as computer programmers usually define it. The definition of a tree used by SQL*Plus is actually somewhat more complex. We'll return to this in a later topic.

A tree has one node which has no owner, and which —directly or indirectly —owns all the other nodes in the tree. The ownerless node is called the *root* of the tree. In the organization chart shown in Figure 13-1, King's row is the tree's root.

If one node owns another, the owning node is said to be the *parent* of the owned node. The owned node is said to be the *child* of the owning node. Each node in a tree has exactly one parent (except for the root node, which has none). A node may have any number of children. A node with no children is called a *terminal node*, or *leaf*.

Nodes which have the same parent are called *siblings*.

A node of a tree, together with all of its children, grandchildren, and so forth, constitutes a *branch* of the tree.

Inspecting each node of a tree in turn is called *walking the tree*. In order to walk a tree, we need some systematic way of ensuring that we will inspect each node one, and only one, time. It is customary to walk a tree according to the following rules:

1. Start at the root node.
2. Inspect this node.
3. If this node has any children that have not yet been visited, move to the leftmost unvisited child and go to step 2. Otherwise:
4. If this is the root node, we're done. Otherwise:
5. Return to this node's parent and go to step 3.

If we walk the tree shown in Figure 13-1 according to these rules, we will visit the nodes in the following order: KING, CLARK, MILLER, JONES, SCOTT, ADAMS, FORD, SMITH, BLAKE, ALLEN, WARD, MARTIN, TURNER, and JAMES.

Exercise 13-1

Walk the tree shown in Figure 13-1 by applying the rules described above.

How a Tree Is Represented in a Table

The relationships that connect the nodes in a tree may be represented by relationships among the rows in a table.

For example, if the nodes shown in Figure 13-1 are represented by the rows in the table EMP, the parent-child relationships that define the tree are represented by relationships between values in the columns MGR and EMPNO. The fact that SCOTT reports to JONES (SCOTT's node is a child of JONES's node) is reflected by the fact that JONES's employee number, found in JONES's EMPNO field, is also found in SCOTT's MGR field.

Suppose we want to write a SELECT command that lists the contents of the table EMP in a way that reflects the tree structure represented in Figure 13-1. We must include a clause that defines the relationship between a parent node and a child node, and a clause that identifies the root node.

We could express the relationship that connects nodes in this tree by the following "logical expression":

parent node's EMPNO = child node's MGR

We can put this relationship into a query with the CONNECT BY clause, which looks like this:

```
CONNECT BY   PRIOR EMPNO = MGR
```

This clause says, "if one node's EMPNO equals a second node's MGR, the first node is the parent of the second." You can remember the meaning of the PRIOR operator by imagining that you are walking the tree from the root downward: you will encounter each parent node *prior* to its child.

We identify the root node with the START WITH clause, which looks like this:

```
START WITH   ENAME = 'KING'
```

This clause says, "the root node (the one to *start with*) is the node in which the logical expression ENAME = 'KING' is true."

Both the CONNECT BY and START WITH clauses must follow the FROM clause (and the WHERE clause, if present).

Exercise 13-2

To list information about employees and their managers in tree form, enter:

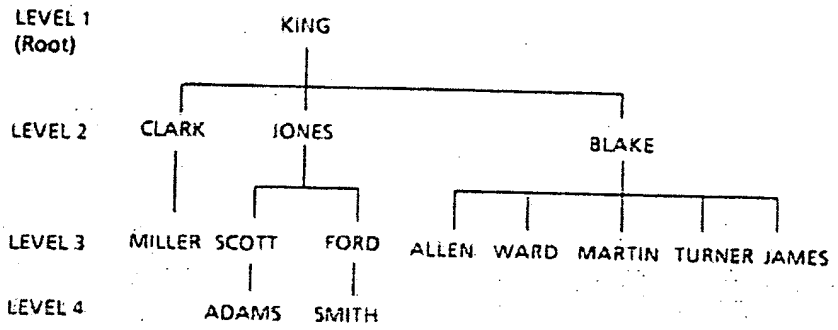
```
SQL> SELECT     ENAME, EMPNO, JOB, DEPTNO, MGR
  2  FROM       EMP
  3  CONNECT BY PRIOR EMPNO = MGR
  4  START WITH ENAME = 'KING';

ENAME      EMPNO JOB       DEPTNO    MGR
---------- ----- --------- ------ ------
KING        7839 PRESIDENT     10
JONES       7566 MANAGER       20   7839
FORD        7902 ANALYST       20   7566
SMITH       7369 CLERK         20   7902
SCOTT       7788 ANALYST       20   7566
ADAMS       7876 CLERK         20   7788
CLARK       7782 MANAGER       10   7839
MILLER      7934 CLERK         10   7782
BLAKE       7698 MANAGER       30   7839
WARD        7521 SALESMAN      30   7698
MARTIN      7654 SALESMAN      30   7698
TURNER      7844 SALESMAN      30   7698
JAMES       7900 CLERK         30   7698
ALLEN       7499 SALESMAN      30   7698

14 records selected.
```

This query returns the rows of EMP by walking the tree from the specified root. King's row, the root, is returned first; then the row for Jones, who works for King; then the row for Ford, who works for Jones; and so on.

Level Numbers

The nodes in a tree may be assigned *level numbers* depending on how far removed they are from the root. The root is level 1, children of the root are level 2, grandchildren are level 3, and so forth. Figure 13-2 shows the levels of the nodes in our sample company's organization chart.

Figure 13-2
Levels of the Organization Chart



LEVEL is a pseudo-column, like SYSDATE (introduced in Chapter 10). You may use LEVEL in an ORDER BY or GROUP BY clause, or anywhere else a column name may appear.

Exercise 13-3

To retrieve level numbers of rows in EMP, enter:

```
SQL> SELECT     LEVEL, ENAME, EMPNO, JOB, DEPTNO, MGR
  2  FROM       EMP
  3  CONNECT BY PRIOR EMPNO = MGR
  4  START WITH ENAME = 'KING';

LEVEL ENAME      EMPNO JOB       DEPTNO    MGR
----- ---------- ----- --------- ------ ------
    1 KING        7839 PRESIDENT     10
    2 JONES       7566 MANAGER       20   7839
    3 SCOTT       7788 ANALYST       20   7566
    4 ADAMS       7876 CLERK         20   7788
    3 FORD        7902 ANALYST       20   7566
    4 SMITH       7369 CLERK         20   7902
    2 CLARK       7782 MANAGER       10   7839
    3 MILLER      7934 CLERK         10   7782
    2 BLAKE       7698 MANAGER       30   7839
    3 ALLEN       7499 SALESMAN      30   7698
    3 WARD        7521 SALESMAN      30   7698
    3 MARTIN      7654 SALESMAN      30   7698
    3 TURNER      7844 SALESMAN      30   7698
    3 JAMES       7900 CLERK         30   7698

14 records selected.
```

You can use the function LPAD to display the organization chart as an indented list.

Exercise 13-4

To create an indented list of employees in manager-employee order, enter:

```
SQL> COLUMN ORG_CHART FORMAT A21;
SQL> SELECT       LPAD(' ',2*LEVEL) || ENAME ORG_CHART, LEVEL, EMPNO,
  2               MGR, DEPTNO
  3  FROM         EMP
  4  CONNECT BY   PRIOR EMPNO = MGR
  5  START WITH   ENAME = 'KING';

ORG_CHART               LEVEL  EMPNO    MGR  DEPTNO
--------------------  -------  -----  -----  ------
KING                        1   7839             10
  JONES                     2   7566   7839      20
    SCOTT                   3   7788   7566      20
      ADAMS                 4   7876   7788      20
    FORD                    3   7902   7566      20
      SMITH                 4   7369   7902      20
  CLARK                     2   7782   7839      10
    MILLER                  3   7934   7782      10
  BLAKE                     2   7698   7839      30
    ALLEN                   3   7499   7698      30
    WARD                    3   7521   7698      30
    MARTIN                  3   7654   7698      30
    TURNER                  3   7844   7698      30
    JAMES                   3   7900   7698      30

14 records selected.
```

In this exercise, the alias ORG_CHART gives the first column its heading. The COLUMN command makes the column 21 characters wide.

In line 1 of the SELECT command, the value of LEVEL is multiplied by 2 (2*LEVEL) to compute the number of blank spaces to precede ENAME.

You can find the average salary at each level of the employee tree by using the pseudo-column LEVEL in the GROUP BY and ORDER BY clauses:

Exercise 13-5

To find the average salary at each level of the employee tree, enter:

```
SQL> SELECT       LEVEL, AVG(SAL)
  2  FROM         EMP
  3  CONNECT BY   PRIOR EMPNO = MGR
  4  START WITH   ENAME = 'KING'
  5  GROUP BY     LEVEL
  6  ORDER BY     LEVEL;

     LEVEL AVG(SAL)
---------- --------
         1     5000
         2  2758.33
         3  1731.25
         4      350
```

SPECIFYING THE ROOT: THE START WITH CLAUSE

The START WITH clause specifies a condition that identifies the root of the tree: ENAME = 'KING' in the preceding exercises. Since everyone in the EMP table works either directly or indirectly for KING (the president), these queries incorporate all of the rows of the table into the organization tree.

But you can identify any other row of the table as the root, and retrieve only the branch of the tree that begins at that row. Suppose you want to know which employees work (directly or indirectly) for JONES:

Exercise 13-6

To list information about employees who work for JONES, enter:

```
SQL> SELECT      LPAD(' ',2*LEVEL) || ENAME (ORG_CHART, LEVEL, EMPNO,
  2              MGR, DEPTNO
  3  FROM        EMP
  4  CONNECT BY  PRIOR EMPNO = MGR
  5  START WITH  ENAME = 'JONES';

ORG_CHART              LEVEL    EMPNO     MGR  DEPTNO
---------------------- -------- ------- ------- -------
JONES                         1    7566   78319      20
SCOTT                         2    7788   75616      20
ADAMS                         3    7876   77818      20
FORD                          2    7902   75616      20
SMITH                         3    7369   79012      20
```

Notice that the LEVEL numbers always begin with 1 at the root specified by START WITH. If you start with JONES, JONES is at level 1, as shown in Figure 13-3.

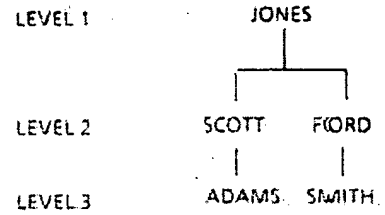


Figure 13-3
A Partial Organization Chart Starting With Jones

The START WITH clause can contain any kind of logical expression. For example, it can construct several trees at once by specifying an expression that is satisfied by more than one row, and so identifies more than one root.

Exercise 13-7

To list information about the people who work for CLARK or BLAKE, enter:

```
SQL> SELECT     LPAD(' ',2*LEVEL) || ENAME ORG_CHART, LEVEL, EMPNO,
  2             MGR, JOB, DEPTNO
  3  FROM       EMP
  4  CONNECT BY PRIOR EMPNO = MGR
  5  START WITH ENAME = 'CLARK'
  6    OR       ENAME = 'BLAKE'
  7  ORDER BY   DEPTNO;

ORG_CHART             LEVEL  EMPNO  MGR JOB       DEPTNO
-------------------- ------ ------ ---- --------- ------
CLARK                     1   7782 7839 MANAGER       10
MILLER                    2   7934 7782 CLERK         10
BLAKE                     1   7698 7839 MANAGER       30
MARTIN                    2   7654 7698 SALESMAN      30
JAMES                     2   7900 7698 CLERK         30
ALLEN                     2   7499 7698 SALESMAN      30
TURNER                    2   7844 7698 SALESMAN      30
WARD                      2   7521 7698 SALESMAN      30

8 records selected.
```

SELECTING A DIRECTION: THE PRIOR OPERATOR

A query may walk the tree either down from root to leaves, or up from leaves to root. You specify the direction of the walk with the PRIOR operator in the CONNECT BY clause.

In the preceding exercises, PRIOR was placed before the parent-node expression in the CONNECT BY clause, making the query walk the tree from the root down to the leaves. If you place PRIOR before the child-node expression, the query walks the tree from one or more leaves up to the root. (Note that this kind of walk will *not* visit all the nodes in the tree unless it identifies every leaf as a starting point.)

Exercise 13-8

To list information about all the people in the organization above SMITH, enter:

```
SQL> SELECT     LPAD(' ',2*LEVEL) || ENAME ORG_CHART, EMPNO, MGR,
  2             JOB, DEPTNO
  3  FROM       EMP
  4  CONNECT BY EMPNO = PRIOR MGR
  5  START WITH ENAME = 'SMITH';

ORG_CHART             EMPNO  MGR JOB       DEPTNO
-------------------- ------ ---- --------- ------
SMITH                  7369 7902 CLERK         20
FORD                   7902 7566 ANALYST       20
JONES                  7566 7839 MANAGER       20
KING                   7839      PRESIDENT     10
```

SELECTING ROWS

As usual, you can use logical expressions to control which rows of a tree are displayed. When a query is tree-structured, though, you can put a logical expression in either the WHERE clause or the CONNECT BY clause.

- The WHERE clause "prunes" individual leaves of the tree.
- The CONNECT BY clause "prunes" entire branches of the tree.

Suppose you want a list of all the people who work for JONES, *except for* SCOTT.

Exercise 13-9

To list information about all the people who work for JONES except SCOTT, enter:

```
SQL> SELECT      ENAME, EMPNO, MGR, JOB, DEPTNO
  2  FROM        EMP
  3  WHERE       ENAME != 'SCOTT'
  4  CONNECT BY  PRIOR EMPNO = MGR
  5  START WITH  ENAME = 'JONES';

ENAME        EMPNO     MGR JOB         DEPTNO
---------- ------- ------- --------- -------
JONES         7566    7839 MANAGER         20
ADAMS         7876    7788 CLERK           20
FORD          7902    7566 ANALYST         20
SMITH         7369    7902 CLERK           20
```

Since the WHERE clause contains the expression ENAME != 'SCOTT', the query eliminates SCOTT from the result. ADAMS, who works for SCOTT, is not eliminated. In other words, the WHERE clause excludes SCOTT's row, but not the branch that "hangs" from it.

To eliminate that branch entirely, use the expression in the CONNECT BY clause:

Exercise 13-10

To list information about all the employees who work for JONES, except SCOTT and those who work for SCOTT, enter:

```
SQL> SELECT      ENAME, EMPNO, MGR, JOB, DEPTNO
  2  FROM        EMP
  3  CONNECT BY  PRIOR EMPNO = MGR
  4    AND       ENAME != 'SCOTT'
  5  START WITH  ENAME = 'JONES';

ENAME        EMPNO      MGR JOB         DEPTNO
---------- ------- ------- --------- -------
JONES         7566    7839 MANAGER        20
FORD          7902    7566 ANALYST        20
SMITH         7369    7902 CLERK          20
```

You may use logical expressions in both clauses to exclude both branches and individual nodes.

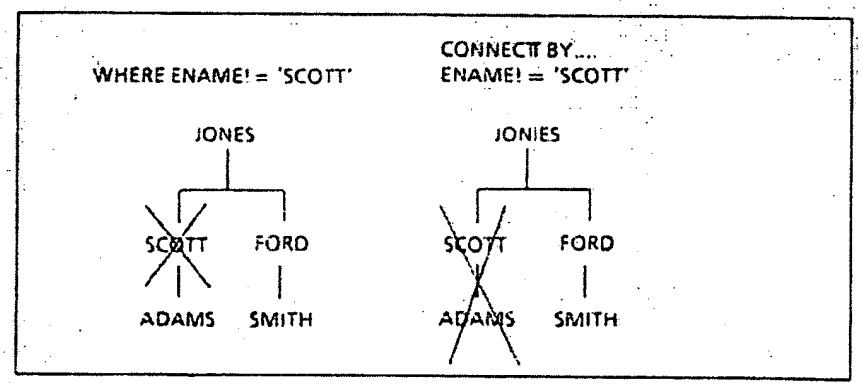


Figure 13-4
Effects of Logical Expressions in WHERE and CONNECT BY Clauses

Exercise 13-11

To list all the employees who work for JONES except FORD, SCOTT, and the people who work for SCOTT, enter:

```
SQL> SELECT      ENAME, EMPNO, MGR, JOB, DEPTNO
  2  FROM        EMP
  3  WHERE       ENAME != 'FORD'
  4  CONNECT BY  PRIOR EMPNO = MGR
  5    AND       ENAME != 'SCOTT'
  6  START WITH  ENAME = 'JONES';

ENAME      EMPNO MGR JOB       DEPTNO
---------- ----- --- --------- ------
JONES       7566 7839 MANAGER       20
SMITH       7369 7902 CLERK         20
```

If you specify conditions with both WHERE and CONNECT BY, the query:

1. Locates the root with the START WITH clause
2. Constructs the tree according to the relationship defined in the CONNECT BY clause
3. Walks the tree in the direction specified by PRIOR
4. Excludes branches of the tree by applying the conditions in the CONNECT BY clause
5. Excludes individual rows by applying the conditions in the WHERE clause to each row
6. Sorts the rows according to the ORDER BY clause

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☑ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.